Dynamical algorithm to balance the load by means of use of vectors of probabilities and adaptative matrixes.

Ángel Azpicueta González[1], Juan Antonio Rodrigo Yanes[2], María del Carmen Fernández Rodríguez[2] y Juan José Pérez[3]

[1] Dep. Automática, Esc. Politécnica Suoerior, Univ. Alcalá
Alcalá de Henares, 28805 Madrid, Spain
angelazpicueta@yahoo.es

[2] Dep. Automática, Esc. Politécnica Suoerior, Univ. Alcalá
Alcalá de Henares, 28805 Madrid, Spain
jrodrigo@aut.uah.es

[3] Dep. Ciencias Computación, Esc. Politécnica Suoerior, Univ. Alcalá
Alcalá de Henares, 28805 Madrid, Spain
juanjose.perez@uah.es

Abstract: In the context of ubiquitous computation, getting the load balanced between the processing units is of relevant importance. In this study a dynamic algorithm is designed in oreder to balance the load of a computer set. A stepwise automaton selectswhich node is the most suitable one to send the required task through a probability vector (namely node to each one of the processing units of the system). This vector is obtained from a dynamical matrix being updated with the information received from the different nodes. As a result, the average computing times concerning the node-executed task are significantly reduced. Also, the algorithm allows to minimize the number of transmitted tasks, avoiding a network overload.

Algoritmo dinámico para equilibrar la carga mediante uso de vectores de probabilidades y matrices adaptativas.

Ángel Azpicueta González[1], Juan Antonio Rodrigo Yanes[2], María del Carmen Fernández Rodríguez[2] y Juan José Pérez[3]

[1] Dep. Automática, Esc. Politécnica Suoerior, Univ. Alcalá
Alcalá de Henares, 28805 Madrid, España
angelazpicueta@yahoo.es
[2] Dep. Automática, Esc. Politécnica Suoerior, Univ. Alcalá
Alcalá de Henares, 28805 Madrid, España
jrodrigo@aut.uah.es
[3] Dep. Ciencias Computación, Esc. Politécnica Suoerior, Univ. Alcalá
Alcalá de Henares, 28805 Madrid, España
juanjose.perez@uah.es

Abstract. El un entorno de computación ubicua es importante conseguir que la carga entre las unidades de procesamiento se encuentre equilibrada correctamente. En este trabajo se propone un algoritmo dinámico para realizar el balanceo de la carga en un conjunto de ordenadores, con el objetivo final de mejorar los tiempos medios de espera de los trabajos que se ejecutan en los distintos nodos (entendiendo por nodo cada una de las unidades de proceso del sistema) y, a la vez, minimizar el tráfico de tareas que se generan para evitar una sobrecarga de la red. Para ello se utiliza un autómata de aprendizaje que, por medio de un vector de probabilidades, decide cual es el nodo idóneo donde enviar una tarea. La actualización de este vector se realiza mediante una matriz dinámica que se actualiza con las informaciones enviadas por los distintos nodos del sistema.

1. Introducción

El equilibrado de la carga es un problema fundamental cuando se trabaja en sistemas distribuidos y paralelos. Gracias a los algoritmos de balanceo de carga se puede conseguir que el rendimiento global del sistema se vea incrementado considerablemente debido a la redistribución de las tareas de los nodos sobrecargados a aquellos que se encuentran ociosos o con una carga ligera. Hay que tener en cuenta que esta migración de tareas entre ordenadores incrementará considerablemente el tráfico de la red de interconexión, y que una migración excesiva de tareas podrá provocar incluso que el rendimiento del sistema se reduzca respecto al caso de no utilizar ningún algoritmo de equilibrado.

Livny y Nelman [1] demostraron que en un sistema de ordenadores homogéneo distribuido, donde todos los nodos tienen la misma potencia de cómputo y reciben el mismo número de tareas, las fluctuaciones estadísticas y los tiempos de

ejecución de las tareas propician una alta probabilidad de que al menos un nodo se encuentre ocioso mientras que, en otro nodo del sistema, haya trabajos esperando para ser procesados.

2. Preliminares

El problema de equilibrar la carga se encuentra íntimamente relacionado con la planificación y asignación de recursos en el sistema. El método de equilibrado puede ser estático o dinámico. En los algoritmos de balanceo estáticos [4] las decisiones se toman en tiempo de compilación, basándose en las estimaciones de los requerimientos de recursos. Sin embargo en los algoritmos dinámicos de balanceo de carga [6],[8],[9],[10],[12],13] la asignación (o re-asignación) de recursos se realiza en tiempo de ejecución basándose en varios parámetros del sistema, los cuales determinan la migración de las tareas entre los nodos, así como las posibles sobrecargas que se puedan generar en el sistema.

Se ha demostrado en numerosos estudios [2],[5] que las técnicas dinámicas ofrecen un rendimiento superior a las estáticas, ya que se adaptan a la evolución del sistema en tiempo de ejecución, cosa que no ocurre con los algoritmos estáticos, ya que en tiempo de compilación no se pueden conocer los recursos necesarios, debido a la inherente aleatoriedad de las tareas en los sistemas reales, en cuanto a tamaño y a tiempo entre recepciones se refiere.

Por lo tanto, la principal ventaja de los algoritmos adaptativos respecto a los estáticos es la inherente flexibilidad de los primeros para adaptarse a los requisitos del sistema en ejecución, a costa de la sobrecarga de comunicaciones entre los nodos. Consecuentemente, las desventajas vienen producidas principalmente por los retardos en las comunicaciones, las transferencias de la información sobre la carga de los nodos y las sobrecargas en cada nodo producidas por el tiempo de procesamiento correspondiente a la toma de decisiones.

2.1 Topología de la red

Otro aspecto importante a tener en cuenta es la topología de la red donde se aplica el sistema de equilibrado de la carga, pudiendo dividirse las redes de interconexión en dos amplias clases: punto a punto (o estáticas) y basadas en conmutador (o dinámicas). Las redes estáticas han sido empleadas en diversas ocasiones como base para equilibrar la carga, pero al estar formadas por enlaces dedicados entre nodos, un elemento de procesamiento no puede comunicarse directamente con todos su vecinos [11],[13],[14]. En la actualidad las redes basadas en conmutador están adquiriendo gran notoriedad [3],[13],[15],[16],[17] debido a que no tienen esta limitación y consiguen unas velocidades de transmisión entre nodos notablemente altas. Si tenemos en cuenta esto, los retardos en la comunicaciones y el tráfico debido a las transferencias de información entre nodos, así como las reasignaciones de tareas entre nodos, dejan de ser un factor decisivo en contra de los algoritmos adaptativos. Es por esto que, aunque se siguen empleando redes punto a punto para balance de carga sobre topologías en árbol, hipercubo, etc., las topologías sobre las que más se aplican los entornos con equilibrado de la carga tienen como base un sistema conmutado.

2.2 Políticas y toma de decisiones en el equilibrado de la carga

La definición de un algoritmo de balance de carga típico consta de tres políticas inherentes bien diferenciadas: la política de información, que especifica la cantidad de información necesaria para tomar las decisiones de reubicación de tareas; la política de transferencia, que especifica las condiciones bajo las cuales se transfiere una tarea, por ejemplo, el número de tareas de un nodo en un momento determinado o el tamaño de las mismas; y la política de localización, que engloba el estudio necesario para la asignación de tareas a los diferentes elementos de procesamiento.

Las diferentes operaciones de equilibrado de la carga se pueden llevar a cabo de manera centralizada en un sólo elemento de proceso, o de manera distribuida entre todos los diferentes nodos de la red. En un entorno distribuido cada elemento de proceso tiene su propia imagen de la carga del sistema en cada momento.

3. Descripción del algoritmo propuesto

El algoritmo de equilibrado de carga que se estudia se basa en un sistema de balanceo descentralizado en un entorno distribuido de ordenadores, los cuales se encuentran completamente interconectados entre si y cuya comunicación se realiza mediante paso de mensajes. Se asume que la red es homogénea y que cualquier tarea puede ser procesada por cualquiera de los nodos del sistema. Todos los elementos de proceso ejecutan la misma imagen de código de la aplicación, y los datos a procesar son divisibles en tareas más pequeñas. Todas las tareas son intensivas en cómputo. Debido a la naturaleza impredecible de las necesidades de recursos de las tareas, no se ha utilizado el tiempo de cómputo como estimación de la carga. En vez de ello, se toma como medida de la carga de los nodos del sistema el número de tareas en la cola de procesos de cada nodo.

Para conseguir estos objetivos el módulo de control del algoritmo estará basado en un autómata de aprendizaje.

3.1 Descripción del autómata de aprendizaje

El autómata de aprendizaje que se utiliza en el algoritmo es el encargado de decidir cual es el nodo que, en el momento de la planificación, tiene una mayor probabilidad de encontrarse desocupado. Esta operación se realiza por medio de un vector de probabilidades que se actualiza en base a una matriz de coeficientes. A su vez la modificación de esta matriz viene determinada por la información proveniente de los nodos a los que se han enviado tareas con anterioridad.

Descripción y análisis del autómata de aprendizaje.

La figura 1 describe el funcionamiento del autómata:

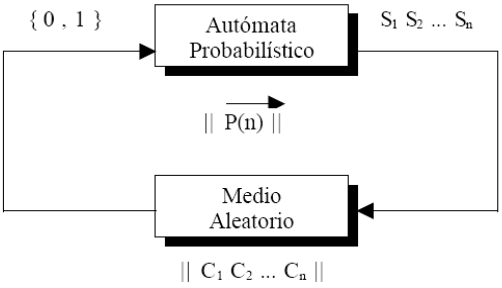

Fig. 1. Esquema del autómata de aprendizaje.

Un autómata de aprendizaje queda descrito por la séxtupla <E,S,Φ,G, P, A> constituida por los elementos descritos a continuación:

- E : $\{e_1, e_2, ... e_e\}$: Conjunto finito de entradas del autómata. Será el conjunto binario (0,1).
- S : $\{ s_1, s_2, ... s_s\}$: Conjunto de salidas o acciones del autómata.
- Φ : $\{ \Phi, \Phi_2, ... \Phi_\Phi\}$: Conjunto finito de estados internos del autómata.
- G: Φ-> S : Aplicación del conjunto Φ sobre S. No se pierde generalidad si consideramos que es biyectiva, es decir, que los conjuntos Φ y S son equivalentes.
- $P(n) = || p_1(n) p_2(n) ... p_p(n) ||$: Vector de probabilidad de estados.
- A : Algoritmo de aprendizaje del autómata. Responsable de la variación de las probabilidades de estados en función del comportamiento global del medio.

Las salidas del autómata vienen determinadas por lo siguiente: Si un nodo recibe una tarea y el tamaño de la cola local no es cero, devolverá al nodo origen de la tarea un castigo (s_1=0), y en caso contrario un premio (s_2=1). Estas salidas se realimentarán en la entrada del nodo que envió la tarea. Por medio de los consecutivos castigos y premios que se van recibiendo en cada nodo, el autómata modifica la matriz de coeficientes y el vector de probabilidades para ser capaz de escoger el nodo que tenga la más alta probabilidad de encontrarse ocioso en el momento en que se realice la siguiente planificación.

De este modo, cuando un nodo recibe una tarea se comprobará si se ha generado localmente o si llega desde un nodo remoto. Si la tarea es local, se analiza el tamaño de la cola; si el nodo no se encuentra cargado, la tarea se deposita en la cola para ser ejecutada. En el caso de que el nodo cumpla la condición de sobrecarga, el algoritmo actualiza el vector de probabilidades y decide a qué nodo se debe enviar la tarea. Entonces, la tarea se envía, y el nodo recoge la respuesta (premio o castigo) del nodo remoto. Si la tarea entrante se ha generado localmente, se pueden estudiar dos casos: depositar la tarea directamente en la cola local, o pasar la tarea al controlador (el sistema de decisión del autómata) para que éste determine si es mejor enviarla a otro nodo menos cargado o ejecutarla localmente.

Método de actualización del autómata.

El vector de probabilidades, en el que se basa la decisión de elegir uno u otro nodo al que enviar una tarea, viene determinado por la matriz de coeficientes. Este es el elemento crucial para el buen funcionamiento del algoritmo, y por eso vamos a analizar en profundidad la estructura del algoritmo de actualización. De forma general la estructura podría quedar como sigue:

$$p(n+1) = T[\, p(n), \quad (n), x(n)\,] \tag{1}$$

donde *T* es un operador, *(n)* representa la acción del autómata, y *x(n)* la entrada del autómata en un instante *x*. Si *p(n+1)* es una función lineal del componente *p(n)* podemos llamar al algoritmo de aprendizaje lineal.

Se puede explicar el funcionamiento del autómata de una manera muy simple: Si el controlador (autómata) envía un trabajo a un nodo y recibe una respuesta de premio por parte del nodo al que ha enviado la tarea, la componente de ese nodo en el vector de probabilidades debe ser incrementada y, en consecuencia, decrementadas las de los demás nodos. Si se recibe un castigo, se actúa de manera inversa. Se debe conseguir que la suma de todas las componentes sea constante e igual a 1 en todo momento ($\sum p_i=1$). La explicación de esta normalización se basa en que al ser cumplida, podemos decir que cada nodo compite con los demás, de tal manera que el aumento de la probabilidad de un nodo repercute en el grado de confianza que el autómata tiene sobre los demás nodos.

Para minimizar las expectativas de penalización se puede elegir entre un gran número de algoritmos de actualización de la matriz de coeficientes. La mayor dificultad que aparece al elegir un autómata determinado es el optar por una mayor velocidad de aprendizaje o por una mayor precisión. Estos dos factores están reñidos, pues si aumentamos la velocidad de convergencia la precisión obtenida se verá lógicamente disminuida. Estudiando estos factores, un posible esquema para cumplir las condiciones expresadas anteriormente podría ser el siguiente proceso estocástico:

$$P_i(t+1) = p_i(t) + p_i(t) \cdot \sum_{\substack{j=1 \\ j \neq i}}^{m} a_{ij} p_j(t) \rightarrow 1 \leq i \leq m \tag{2}$$

Este proceso estocástico genera un conjunto de ecuaciones diferenciales no lineales entre probabilidades. El sistema tiene como soluciones extremas (que se alcanzan al llegar a la estabilidad): (1,0,...,0), (0,1,...,0) ... (0,0,...,1).

El conjunto de elementos de la matriz de coeficientes, $A(a_{ij})$ es el siguiente:

$$\begin{pmatrix} 0 & A_{12} & A_{13} & \dots & A_{1m} \\ A_{21} & 0 & A_{23} & \dots & A_{2m} \\ \dots & \dots & \dots & \dots & \\ A_{m1} & A_{m2} & A_{m3} & \dots & 0 \end{pmatrix}$$

En esta matriz se cumple siempre la siguiente condición: $A_{ij} = -A_{j\,i}$.

Para establecer un algoritmo para la actualización del estado de probabilidades es esencial la convergencia incondicional del proceso estocástico de la ecuación (2) a una de las soluciones extremas. Por lo tanto se intentará que se castigue lo menos posible al nodo cuya probabilidad sea 1, intentando adaptar el estado de probabilidades del autómata a la ecuación anterior. Para lograr esto y teniendo en cuenta que los coeficientes de la matriz controlan la convergencia, la idea básica es tan simple como que si un autómata selecciona un nodo, le envía una tarea y recibe premio, las componentes de la fila correspondiente al nodo son incrementadas, mientras que se decrementan la componentes correspondientes a la columna que marca ese nodo. Si la respuesta del nodo es un castigo, se opera de manera inversa.
Siguiendo esto, las probabilidades futuras serán establecidas siguiendo las ecuaciones que se expresan a continuación:

$$p_i(t+1) = p_i(t) + \sum_{\substack{j=1 \\ j \neq i}}^{m} (a_{ij} + \Delta a_{ij}) \cdot p_i(t) \cdot p_j(t) \qquad (3)$$

$$p_h(t+1) = p_h(t) + \sum_{\substack{j=1 \\ j \neq i,h}}^{m} a_{hj} p_h(t) p_j(t) + (a_{hi} - \Delta a_{ih}) \cdot p_h(t) \cdot p_i(t) \rightarrow 1 \leq h \leq m, h \neq i \qquad (4)$$

Estas ecuaciones corresponden a la recepción de premios de los demás nodos. Si en la recepción aparece un castigo, los signos correspondientes a los incrementos se deberían cambiar para convertirlos en decrementos.

3.2 Descripción de los algoritmos utilizados en las simulaciones

Para poder obtener los valores idóneos de los diferentes parámetros de ajuste para el algoritmo se han realizado simulaciones modificando diversas variables como la carga de sistema, los valores máximos de los elementos de la matriz de coeficientes, el incremento de estos cada vez que se realiza una planificación y el tamaño de la cola para el que debe entrar en funcionamiento el controlador. Así, se han simulado tres distintas variaciones con distintos parámetros:

- Variación 1: Los elementos de la matriz de coeficientes varían entre -1 y 1, pero no pueden ser iguales a estos límites.
- Variación 2: Los valores de la matriz de coeficientes varían entre -1 y 1, pudiendo llegar a alcanzar dichos valores.
- Variación 3: Igual que la variación 2, pero pasando todas las tareas entrantes al controlador, ya sean generadas localmente o recibidas de otros nodos.

Las tres variaciones se han simulado con distintos tipos de carga (ligera y moderada), y con distinto umbral en el tamaño de las colas para iniciar el balance.

4. Análisis de resultados

La simulaciones que han arrojado los datos que se presentan a continuación se han

realizado sobre una topología de red basada en conmutación y con 5 máquinas iguales. El código que se ejecutaba en cada simulación era el mismo para cada nodo.

Comenzando con la simulación de la primera variación, los parámetros que se han modificado han sido el incremento de los elementos de la matriz (0.1 - 0.4), y el tamaño de la cola local a partir del cual se ejecuta el controlador (2-4).

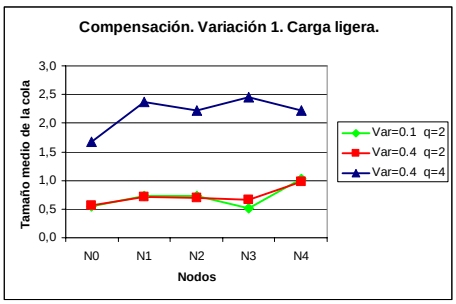

Fig. 2. Compensación, Variación 1, carga ligera.

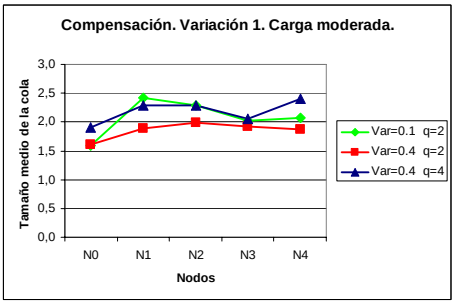

Fig. 3. Compensación. Variación 1, carga moderada

En las figuras anteriores podemos observar el comportamiento del sistema para cargas moderada y ligeras. Al trabajar con cargas ligeras, la compensación de la carga entre los nodos es lógicamente inferior, ya que al superarse en menos ocasiones el umbral para la activación del controlador, el movimiento de tareas entre nodos se reduce. Con este tipo de carga ya podemos observar que la mejor compensación se obtiene al fijar las variaciones de los elementos de la matriz en un valor Δ=0.4 y establecer el umbral de la cola de tareas en q=2. Si observamos el comportamiento del sistema cuando se le aplica una carga moderada, podemos comprobar que la compensación es mejor en los 3 casos y que, además, los valores que mejor lo consiguen son los mismos que con carga ligera, e incluso reducen el número medio de clientes en la cola. Por lo tanto elegiremos q=2 como valor umbral para iniciar el proceso de balance de la carga entre los nodos.

Al simular la segunda variación del algoritmo, donde los elementos de la matriz pueden alcanzar los valores '-1' y '1', podemos incluir como incremento Δ=0.5 (en la variación anterior no era útil, pues los elementos de la matriz sólo hubieran podido tomar 3 valores: -0.5, 0.0 y 0.5, y la estabilidad del algoritmo hubiera sido muy baja). Por lo tanto, teniendo en cuenta que el mejor funcionamiento de la primera variación del algoritmo se ha obtenido con un valor Δ=0.4, comparamos la compensación obtenida anteriormente con la resultante de la simulación de la segunda variación cambiando el incremento de los elementos de la matriz.

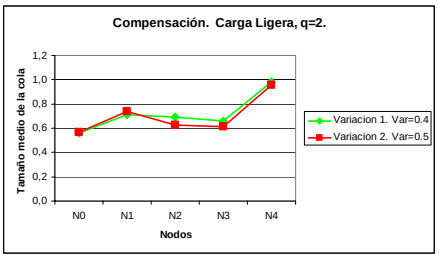

Fig. 4. Compensación. Carga ligera, *q*=2.

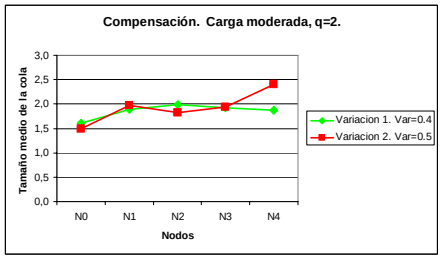

Fig. 5. Compensación. Carga moderada, *q*=2.

Podemos observar en las figuras 4 y 5 que la compensación que se realiza al trabajar con cargas ligeras es muy similar con ambos valores, pero al incrementar la carga del sistema, la compensación es mayor con el valor Δ=0.4. Este empeoramiento con el valor 0.5 se debe a la rapidez con la que reacciona el algoritmo (si 2 nodos aprenden a la vez que un tercero es el menos cargado, comenzarán a mandarle rápidamente tareas, con la consiguiente sobrecarga de este nodo). No hay que olvidar que los valores que toman los elementos de la matriz influyen determinantemente en el vector de probabilidades que es el que rige el comportamiento del controlador.

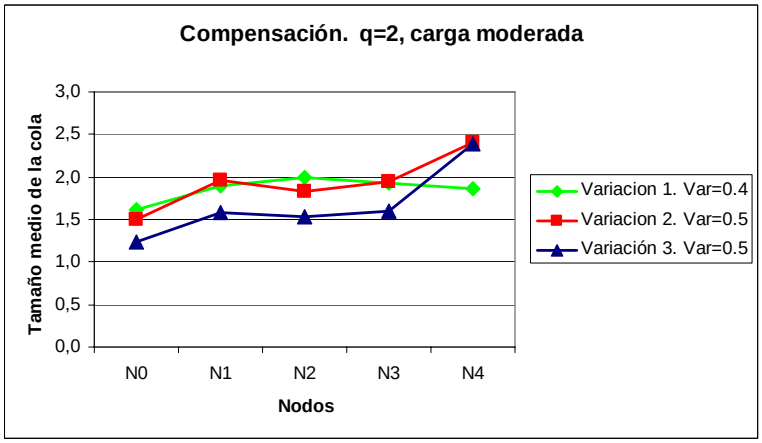

Fig. 6. Compensación. Carga moderada, *q*=2.

Al tomar los datos de la simulación de la tercera variación, donde todas las tareas entrantes a un nodo (generadas localmente y recibidas de otros nodos) se pasan al controlador, y compararlos con los datos de las dos anteriores variaciones, podemos observar (Fig. 6) que la mejor compensación se sigue consiguiendo utilizando la primera variación, que tomaba Δ=0.4 (en vez de Δ=0.5 que utilizan las otras dos variaciones del algoritmo). Como hemos comentado, la rapidez con la varían los elementos de la matriz puede hacer que no se consiga la compensación deseada. Sin embargo, si nos fijamos en la tercera variación, podemos observar que el número medio de tareas en la cola es considerablemente menor con respecto a la primera y

segunda variación. El punto negativo de la tercera variación viene determinado por el aumento del envío de tareas entre nodos, y el consiguiente aumento del tráfico en la red de interconexión. Si tomamos como indicativo el porcentaje de tareas que envía un nodo respecto a las que se crean localmente, podemos observar en la fig. 7 que el número de tareas enviadas por los nodos cuando ejecutan la tercera variación del algoritmo es muy superior al que se genera en los otros dos casos. Esto es debido a que una tarea puede ser reenviada varias veces entre los nodos antes de encontrar un nodo poco cargado que decida depositarla en su cola local para ser ejecutada.

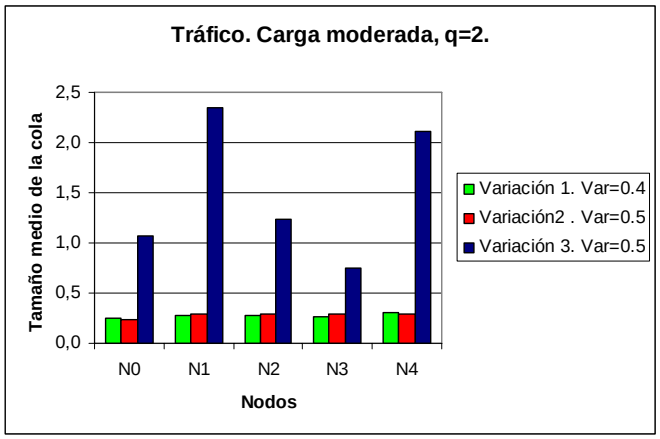

Fig. 7. Compensación. Carga moderada, q=2.

Como resumen podemos decir que la primera variación del algoritmo, donde Δ=0.4, trabajando con un umbral q=2, es la que mejor compensación de carga consigue, manteniendo un tráfico en la red bastante bajo. A la vez, la tercera variación, con Δ=0.5 y q=2, consigue una compensación media y, además, reduce el número medio de tareas en la cola de los nodos, lo que ayuda a reducir el tiempo medio de ejecución de las tareas. Sin embargo, el tráfico que genera en la red es muy elevado, lo que se convierte en un problema a la hora de realizar el equilibrado real de un sistema con este tipo de algoritmo.

5. Conclusiones y trabajo futuro.

El algoritmo propuesto se basa en un autómata de aprendizaje que modifica una matriz de coeficientes basándose en las respuestas que devuelven los distintos nodos que colaboran en la simulación. A partir de esta matriz, podemos conseguir un vector de probabilidades para saber en cada momento cual es el nodo que tienen más probabilidades de estar ocioso. Hemos podido comprobar que las distintas variaciones de este algoritmo funcionan correctamente, pues consiguen una mejor compensación de carga entre los nodos, lo que se traduce, finalmente, en un menor tiempo de servicio para las tareas. Cualquiera de las variaciones representa una ganancia respecto a un entorno en el que los distintos nodos no equilibraran la carga de tareas con los demás.

Se sigue estudiando este tipo de algoritmos estudiando nuevos parámetros para conseguir una mejora de la compensación de carga y del tiempo medio de servicio de las tareas. El siguiente paso será aplicar el autómata de aprendizaje sobre una red de interconexión basada en conmutador, que permita el envío de tareas entre un nodo y cualquiera de sus vecinos. Además, se comparará el comportamiento del algoritmo propuesto con el de algoritmos basados sobre topologías de red punto a punto de más alta velocidad entre nodos, como hipercubos, árboles, etc., simulando virtualmente este tipo de redes en el entorno de trabajo conmutado de nuestro algoritmo. Otro objetivo es unir las posibles ventajas de los algoritmos basados en este tipo de redes con los beneficios, ya demostrados, de trabajar con algoritmos basados en autómatas de aprendizaje sobre redes conmutadas, como es el que se ha propuesto y analizado.

Referencias

1. Livny, M., Melman, M.: Load balancing in homogeneous broadcast distributed systems. In Proceedings of the ACM Computer Network Performance Symposium, pp. 47-55, (April 1982).
2. http://www.loadbalancing.net Online Resource for Load Balancing Information.
3. Azpicueta, A.:Reparto de carga en sistemas distribuidos. Universidad Alcalá de Henares (2000).
4. Shirazi B.A., Hurson, A.R., Kavi, K.M.: Scheduling and Load Balancing on Parallel and Distributed Systems. IEEE Computer Society Press, Los Alamitos, CA (1995).
5. Zaki, M.J., Li, W., Parthasarathy, S.: Customized Dynamic Load Balancing for a Network of Workstations. J. Prallel Distrib. Comput. 43, (1997) 156-162.
6. Cybenko, G.: Dynamic load balancing for distributed-memory multiprocessors. J. Parallel Distrib. Comput. 7 (1989) 279-301.
7. Stankovic, J.A.: Simulations of three adaptative, decentralized controlled, job scheduling algorithms. Computer networks. 8 (1984) 199-217.
8. Eager, D.L., Lazowska, E.D., Zahorjan, J.: Adaptative load sharing in homogeneous distributed systems. IEEE Trans. Software Eng. 12 (1986) 662-675.
9. Fox, G., Kalawa, A., Williams, R.: The implementation of a dynamic load balancer. In Proceedings of the Conference on Hypercube Multiprocessors, pp. (1987) 114-121.
10. Hu, Y.F., Blake, R.J.: An improved diffusion algorithm for dynamic load balancing. Parallel Comput. 25 (1999) 417-444.
11. Rim, H., Jang, J., Kim, S.: A simple reduction of non- uniformity in dynamic load balancing of quantized loads on hypercube multiprocessors and hidding balance overheads. Journal of Computer and System Sciences 67 (2003) 1-25.
12. Antonis, K.,J. Garofalakis, Mourtos, I., Spirakis, P.: A hierarchical adaptative algorithm for load balancing. J. Parallel Distrib. Comput. 64 (2004) 151-162.
13. Lee, W.Y., Hong, S.J., Kim, J., Lee, S.: Dynamic load balancing for switch-based networks. J. Prallel Distrib. Comput. 63 (2003) 286-298.
14. Cortés, A., Ripoll, A., Cedó, F., Senar, M.A., Luque, E.: An asynchronous and iterative load balancing algorithm for discrete load model. J. Parallel Distrib. Comput. 62 (2002) 1729-1746.
15. Das, S.K., Harvey, D.J., Biswas, R.: Adaptative Load-Balancing Algorithms using Symmetric Broadcast Networks. J. Parallel Distrib. Comput. 62 (2002) 1042-1068.
16. Chronopoulus, A.T., Grosu, D., Wissink, A.M., Benche, M., Liu, J.: An efficient 3D based scheduling for heterogeneus systems. J. Parallel Distrib. Comput. 63 (2003) 827-837.
17. Zeng, Z., Veeravalli, B.: Design and analysis of a non-preemptive decentralized load

balancing algorithm for multi-class jobs in distributed networks. Computer Communications 27 (2004) 679-693.